Homework 1: Foundations

Course: CS 221 Spring 2019
Name: Bryan Yaggi

Problem 1: Optimization and Probability

In this class, we will cast a lot of AI problems as optimization problems, that is, finding the best solution in a rigorous mathematical sense. At the same time, we must be adroit at coping with uncertainty in the world, and for that, we appeal to tools from probability.

(a) Let $x_1, , x_n$ be real numbers representing positions on a number line. Let $w_1, , w_n$ be (strictly) positive real numbers representing the importance of each of these positions. Consider the quadratic function: $f(\theta) = \frac{1}{2}\sum_{i=1}^{n} w_i(\theta - x_i)^2$. What value of θ minimizes $f(\theta)$? You can think about this problem as trying to find the point that's not too far away from the x_i's. Over time, hopefully you'll appreciate how nice quadratic functions are to minimize. How will your answer change if some of the w_i's are negative?

The solution can be found by taking the derivative of the function with respect to θ and setting it equal to zero.

$$\begin{aligned}
\frac{\partial f(\theta)}{\partial \theta} &= \frac{1}{2}\sum_{i=1}^{n} 2w_i(\theta - x_i)(1) && \text{by the chain rule}\\
&= \sum_{i=1}^{n} w_i(\theta - x_i)\\
\frac{\partial f(\theta)}{\partial \theta} = 0 \implies 0 &= \sum_{i=1}^{n} w_i\theta - \sum_{i=1}^{n} w_i x_i && \text{setting equal to zero}\\
\sum_{i=1}^{n} w_i x_i &= \sum_{i=1}^{n}\theta\sum_{i=1}^{n} w_i\\
&= n\theta\sum_{i=1}^{n} w_i\\
\theta &= \frac{\sum_{i=1}^{n} x_i}{n}
\end{aligned}$$

If the second derivative of the function is negative, which could happen if some of the w_is are negative, the function will not have a minimum.

(b) In this class, there will be a lot of sums and maxes. Let's see what happens if we switch the order. Let $f(x) = \sum_{i=1}^{d} max_{s\in\{1,-1\}} sx_i$ and $g(x) = max_{s\in\{1,-1\}} \sum_{i=1}^{d} sx_i$, where $x = (x_1, \ldots, x_d) \in \mathbb{R}^d$ is a real vector. Does $f(x) \leq g(x)$, $f(x) = g(x)$, or $f(x) \geq g(x)$ hold for all x? Prove it. Hint: You may find it helpful to refactor the expressions so that they are maximizing the same quantity over different sized sets.

$$f(x) \geq g(x)$$

$f(x)$ takes the maximum of each element before the summation, so each term summed will be positive. $g(x)$ takes the maximum of the summations for the 2 possible values of s. The terms summed might not all be positive. Threfore, $f(x) \geq g(x)$.

(c) Suppose you repeatedly roll a fair six-sided die until you roll a 1 (and then you stop). Every time you roll a 2, you lose a points, and every time you roll a 6, you win b points. You do not win or lose any points if you roll a 3, 4, or a 5. What is the expected number of points (as a function of a and b) you will have when you stop? Hint: it is recommended to think of defining a recurrence.

Really, there are only 3 rolls that affect the number of points earned: rolling a 1, 2, or 6. Each time a non-one number is rolled, the probalility of each significant roll is the same as at the start. The expected number of points can be defined recursively. Let Y be the number of points earned when you stop. Let X be the number rolled.

$$\begin{aligned}
E[Y] &= E[Y|X=1]P(X=1) + E[Y|X=2]P(X=2) + E[Y|X=6]P(X=6) \\
&\quad P(X=1) = P(X=2) = P(X=6) = \frac{1}{3} \\
&\quad E[Y|X=1] = 0 \\
&\quad E[Y|X=2] = E[Y] - a \\
&\quad E[Y|X=6] = E[Y] + b \\
E[Y] &= (0)\frac{1}{3} + (E[Y]-a)\frac{1}{3} + (E[Y]+b)\frac{1}{3} \\
E[Y] &= b - a
\end{aligned}$$

(d) You are playing a game with a fair five-sided die (sides 1,2,3,4,5 and probability $\frac{1}{5}$ of coming up each of these). At each turn, you have an option to quit and win 15 points (and thus roll no more) or to win 3 points and roll the dice. If at any point, you roll an even number, the game ends, and you leave with your total winnings. Else you conitnue onto the next turn. What is the optimal strategy and why? What would change if the dice was weighted so that the probability of rolling an even number was 0?

To find the optimum solution, consider the expected number of points for possible strategies. Let Y be the number of points earned. Let X be the number rolled.

First, consider the strategy where the player quits before rolling.

$$E[Y] = 15$$

Now, consider the strategy where the player does not quit.

$$\begin{aligned}
E[Y] &= E[Y|X \textit{ is odd}]P(X \textit{ is odd}) + E[Y|X \textit{ is even}]P(X \textit{ is even}) \\
E[Y] &= E[Y+3]\frac{3}{5} + (3)\frac{2}{5} \\
E[Y] &= 7.2
\end{aligned}$$

Consider the strategy where the player quits after one roll.

$$\begin{aligned}
E[Y] &= E[Y|X \textit{ is odd}]P(X \textit{ is odd}) + E[Y|X \textit{ is even}]P(X \textit{ is even}) \\
E[Y] &= (18)\frac{3}{5} + (3)\frac{2}{5} \\
E[Y] &= 12
\end{aligned}$$

Now, consider the strategy where the player quits after 2 rolls.

$$\begin{aligned}
E[Y] &= E[Y|X\ is\ even\ on\ first\ roll]P(X\ is\ even\ on\ first\ roll) \\
&\quad + E[Y|X\ is\ even\ on\ first\ roll,\ odd\ on\ second]P(X\ is\ even\ on\ first\ roll,\ odd\ on\ second) \\
&\quad + E[Y|X\ is\ even\ on\ first\ roll,\ even\ on\ second]P(X\ is\ even\ on\ first\ roll,\ even\ on\ second) \\
E[Y] &= (3)\frac{2}{5} + (6)\frac{3}{5}\frac{2}{5} + (21)\frac{3}{5}\frac{3}{5} \\
E[Y] &= 10.2
\end{aligned}$$

It looks like the best strategy is to quit and take the 15 points at the beginning. If the probablility of rolling an even number were zero, it'd be best to roll infinitely (or until you get tired).

(e) Suppose the probability of a coin turning up heads is $0 < p < 1$, and that we flip it 7 times and get $\{H, H, T, H, T, T, H\}$. We know the probability (likelihood) of obtaining this sequence is $L(p) = pp(1-p)p(1-p)(1-p)p = p^4(1-p)^3$. Now let's go back and ask the question: what value of p maximizes $L(p)$? What is an intuitive interpretation of this value of p? Hint: Consider taking the derivative of $logL(p)$. You can also directly take the derivative of $L(p)$, but it is cleaner and more natural to differentiate $logL(p)$. You can verify for yourself that the value of p which maximizes $logL(p)$ must also maximize $L(p)$ (you are not required to prove this in your solution).

$$\begin{aligned}
L(p) &= p^4(1-p)^3 \\
logL(p) &= 4log(p) + 3log(1-p) \\
\frac{\partial logL(p)}{\partial p} &= 4\frac{1}{p} + 3\frac{-1}{1-p} \\
\frac{\partial logL(p)}{\partial p} = 0 \implies 0 &= \frac{4}{p} - \frac{3}{1-p} && \text{setting equal to zero} \\
p &= \frac{4}{7}
\end{aligned}$$

This makes sense because flipping 4 heads is most likely with this probablility.

(f) Let's practice taking gradients, which is a key operation for being able to optimize continuous functions. For $w \in \mathbb{R}^d$ (represented as a column vector) and constants $a_i, b_j \in \mathbb{R}^d$ (also represented as column vectors) and $\lambda \in \mathbb{R}$, define the scalar-valued function

$$f(w) = \sum_{i=1}^{n}\sum_{j=1}^{n}(a_i^\top w - b_j^\top w)^2 + \lambda\|w\|_2^2,$$

where the vector is $w = (w_1, \ldots, w_d)^\top$ and $\|w\|_2 = \sqrt{\sum_{k=1}^{d} w_k^2}$ is known as the L_2 norm. Compute the gradient $\nabla f(w)$. Recall: the gradient is a d-dimensional vector of the partial derivatives with respect to each w_i:

$$\nabla f(w) = (\frac{\partial f(w)}{\partial w_1}, \ldots, \frac{\partial f(w)}{\partial w_d})^\top.$$

If you're not comfortable with vector calculus, first warm up by working out this problem using scalars in place of vectors and derivatives in place of gradients. Not everything for scalars goes through for vectors, but the two should at least be consistent with each other (when $d = 1$). Do not write out summation over dimensions, because that gets tedious.

$$\nabla f(w) = \begin{bmatrix} \frac{\partial f(w)}{\partial w_1} \\ \vdots \\ \frac{\partial f(w)}{\partial w_d} \end{bmatrix} = \begin{bmatrix} 2\sum_{i=1}^n \sum_{j=1}^n ((a_i^\top w_1 + b_j^\top w_1)(a_i + b_j)) + 2\lambda w_1 \\ \vdots \\ 2\sum_{i=1}^n \sum_{j=1}^n ((a_i^\top w_d + b_j^\top w_d)(a_i + b_j)) + 2\lambda w_d \end{bmatrix}$$

Problem 2: Complexity

When designing algorithms, it's useful to be able to do quick back of the envelope calculations to see how much time or space an algorithm needs. Hopefully, you'll start to get more intuition for this by being exposed to different types of problems.

(a) Suppose we have an image of a human face consisting of nxn pixels. In our simplified setting, a face consists of two eyes, two ears, one nose, and one mouth, each represented as an arbitrary axis-aligned rectangle (i.e. the axes of the rectangle are aligned with the axes of the image). As we'd like to handle Picasso portraits too, there are no constraints on the location or size of the rectangles. How many possible faces (choice of its component rectangles) are there? In general, we only care about asymptotic complexity, so give your answer in the form of $O(nc)$ or $O(cn)$ for some integer c.

To define a rectange, two bounding horizontal axes and two bounding vertical axes can be selected. 6 rectangles need to be chosen to constitute a face. This is a combinatorics problem.

$$\left(\binom{n+1}{2}^2\right)^6 = \left(\frac{(n+1)!}{2!(n-1)!}\right)^{12} = \left(\frac{(n+1)(n)}{4}\right)^{12} \implies O(n^{24})$$

(b) Suppose we have an nxn grid. We start in the upper-left corner (position $(1, 1)$), and we would like to reach the lower-right corner (position (n, n)) by taking single steps down and right. Define a function $c(i, j)$ to be the cost of touching position (i, j), and assume it takes constant time to compute. Note that $c(i, j)$ can be negative. Give an algorithm for computing the minimum cost in the most efficient way. What is the runtime (just give the big-O)?

The problem can be solved using recursion and dynamic programming. The complexity is $O(n^2)$. The following is a code sample in Python.

```
cache = {}
def findMinCost(row, col):
        if (row, col) in cache:
                return cache[(row, col)]
        if row == target[0] and col == target[1]:
                result = c(row, col)
        elif row == target[0]:
                result = c(row, col) + findMinCost(row, col+1)
        elif col == target[1]:
                result = c(row, col) + findMinCost(row+1, col)
        else:
                result = c(row, col) + min(findMinCost(row+1, col),
                        findMinCost(row, col+1))

        cache[(row, col)] = result
        return result
```

(c) Suppose we have a staircase with n steps (we start on the ground, so we need n total steps to reach the top). We can take as many steps forward at a time, but we will never step backwards. How many ways are there to reach the top? Give your answer as a function of n. For example, if $n = 3$, then the answer is 4. The four options are the following: (1) take one step, take one step, take one step (2) take two steps, take one step (3) take one step, take two steps (4) take three steps.

The solution was found by experimentation. Let $f(n)$ be the number of ways for a staircase with n stairs.

$$\begin{aligned}
f(1) &= 1 && [1] \\
f(2) &= 2 && [2] \text{ or } [1,1] \\
f(3) &= 4 && [3] \text{ or } [2,1] \text{ or } [1,2] \text{ or } [1,1,1] \\
f(4) &= 8 && [4] \text{ or } [3,1] \text{ or } [1,3] \text{ or } [2,2] \text{ or } [2,1,1] \text{ or } [1,2,1] \text{ or } [1,1,2] \text{ or } [1,1,1,1] \\
f(n) &= 2^{n-1}
\end{aligned}$$

(d) Consider the scalar-valued function $f(w)$ from Problem 1f. Devise a strategy that first does preprocessing in $O(nd^2)$ time, and then for any given vector w, takes $O(d^2)$ time instead to compute $f(w)$. Hint: Refactor the algebraic expression; this is a classic trick used in machine learning. Again, you may find it helpful to work out the scalar case first.

$$\begin{aligned}
f(w) &= \sum_{i=1}^{n}\sum_{j=1}^{n}(a_i^\top w - b_j^\top w)^2 + \lambda\|w\|_2^2 \\
&= w^\top \left(\sum_{i=1}^{n}\sum_{j=1}^{n}(a_i - b_j)(a_i - b_j)^\top\right) w + \lambda w^\top w \\
&= \left(\left(\sum_{i=1}^{n} a_i - \sum_{j=1}^{n} b_j\right)\left(\sum_{i=1}^{n} a_i - \sum_{j=1}^{n} b_j\right)^\top + \lambda\right) w^\top w
\end{aligned}$$